University of Pittsburgh

Algorithms and Data Structures 2
CS 1501

Spring 2022

Sherif Khattab

ksm73@pitt.edu

(Slides are adapted from Dr. Ramirez's and Dr. Farnan's CS1501 slides.)

Announcements

- Upcoming deadlines:
  - Assignment 1 late deadline is tonight 3/16
  - Homework 8 due on 3/21
  - Lab 7 due on 3/18
  - Assignment 2 due on 3/28

Previous Lecture …

- Graphs
  - Definitions, representation, traversal (BFS and DFS)
- Given two LI accounts, how "far" are they from each other?
  - E.g., 1st connection, 2nd connection, etc.
- Are the accounts in the file all ***connected***?
  - If not, how many ***connected components*** are there?

Muddiest Points (confusing)

- I was confused about how many adjacent lines exist between undirected graphs vs directed graphs.
- In the neighbor lists, are directed edges counted once or twice?
- LZW is still a little confusing at times. Is there an easier way to break it down into steps.
- I was confused to how DFS would find the articulation points of a graph
- the DFS and BFS algorithms
- How the queue works for DFS/BFS but I think it'll be more clear once I look over it again/we have the homework on it
- - The different operations that can be performed on a graph was most confusing.

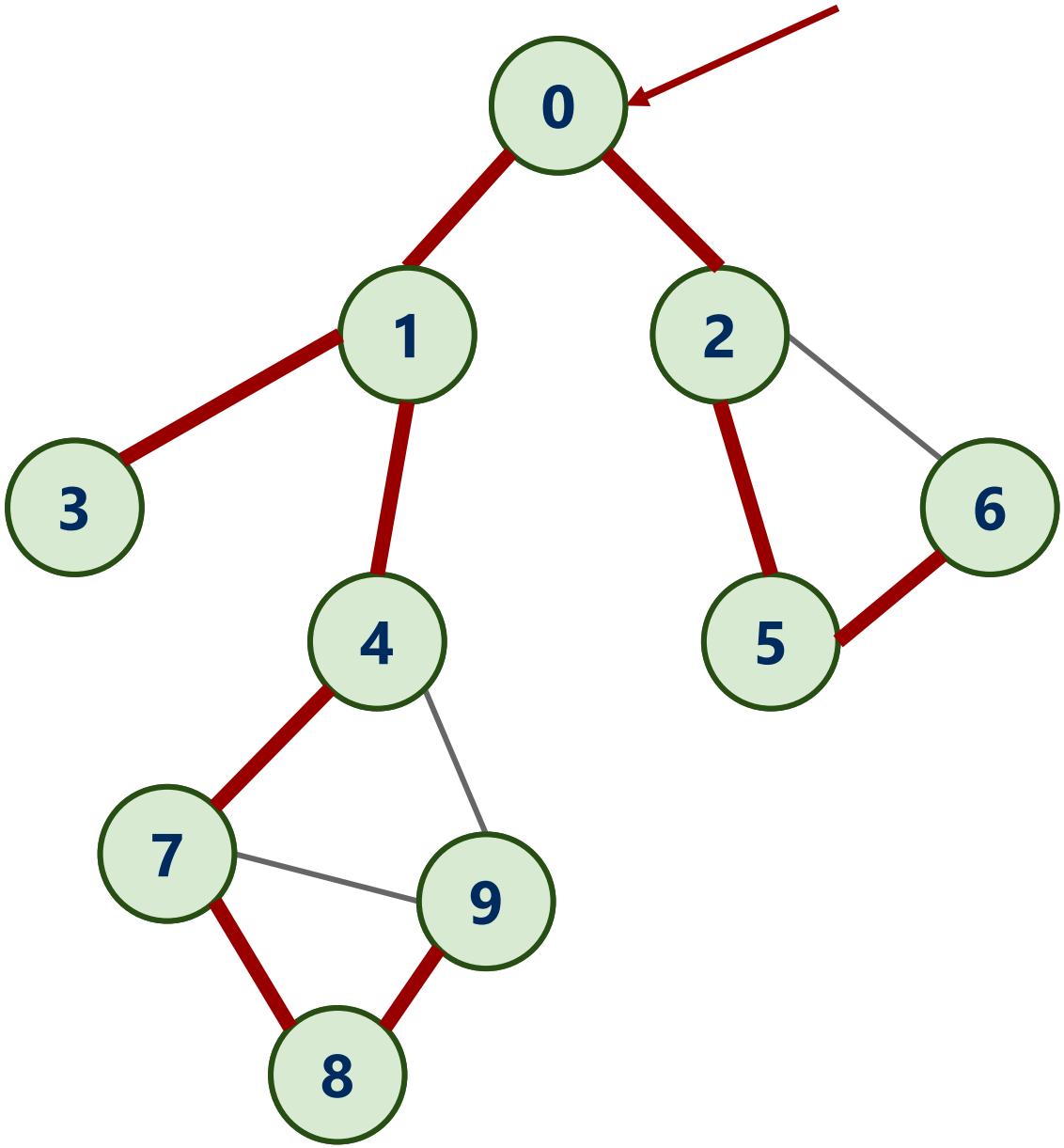
Muddiest Points (interesting)

- Graphs were interesting to think about and reminded me of math concepts
- I found it interesting how going through the searches produced spanning trees
- Top hat questions help a lot with learning the material. Having them through lecture is nice
- Edges , vertices
- That trees are just a special subclass of graphs and that graphs have so many uses in the real world
- runtime of graph search algorithms
- That we can model many different relationships and connections using graphs and graph theory
- I worked with graph theory before for research but it was interesting to see other applications
- The applications of a graph was most interesting.
- the matrix vs array comparison

Problem of the Day (*contd.*)

- **Input**: A file containing LinkedIn Connection information formatted like the following:
  - Account1: Connection1, Connection2, …
  - Account2: Connection1, Connection2, …
  - …
- **Output**: Answer the following questions:
  - Given two LI accounts, how "far" are they from each other?
    - E.g., 1st connection, 2nd connection, etc.
  - Are the accounts in the file all ***connected***?
    - If not, how many ***connected components*** are there?
  - Are there certain accounts that if removed, the remaining accounts become ***partitioned***?
    - These account are called ***articulation points***

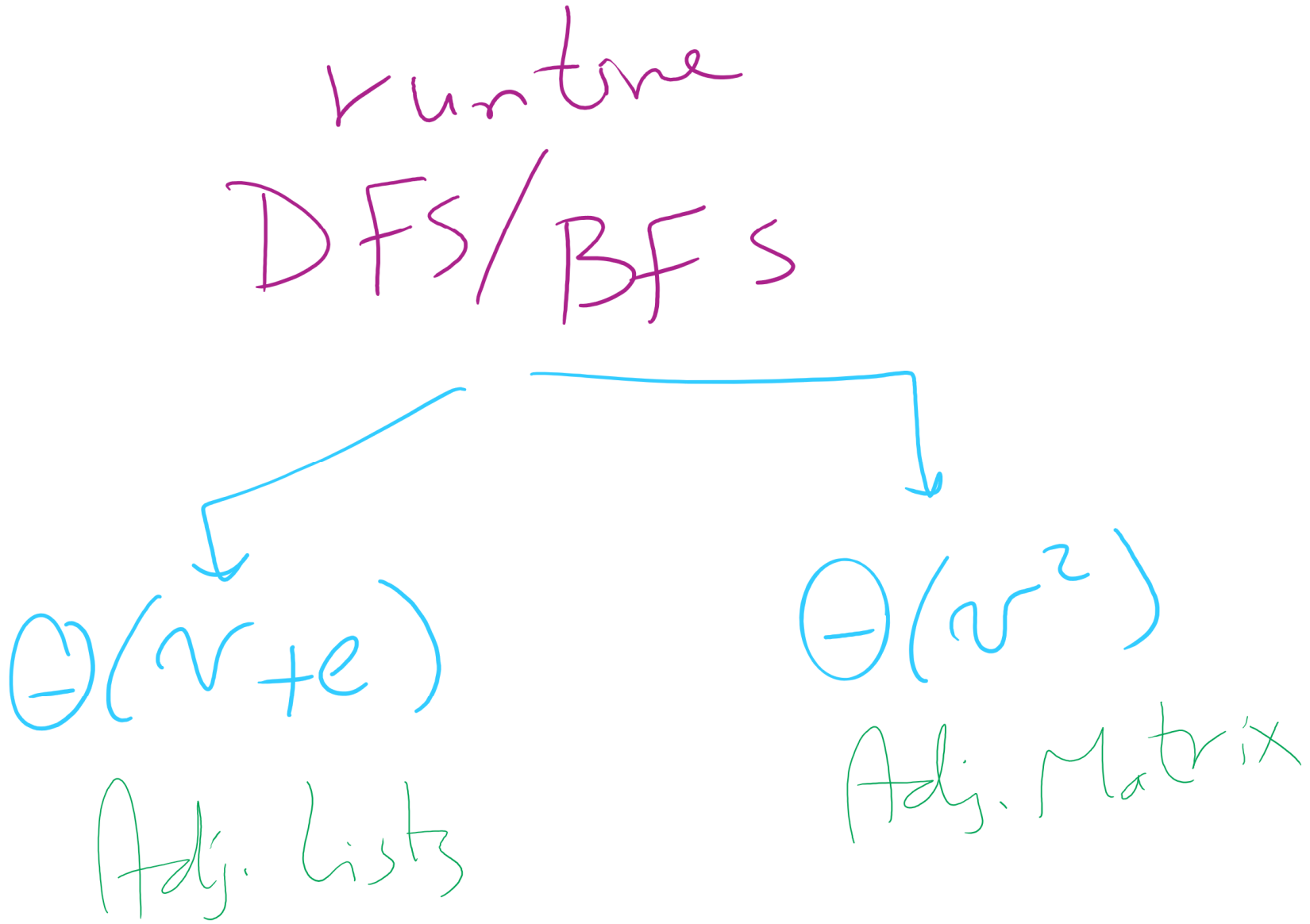
DFS example 2



Analysis of graph traversals

- At a high level, DFS and BFS have the same runtime
  - Each vertex must be seen and then visited, but the order will differ between these two approaches
- How will the representation of the graph affect the runtimes of of these traversal algorithms?

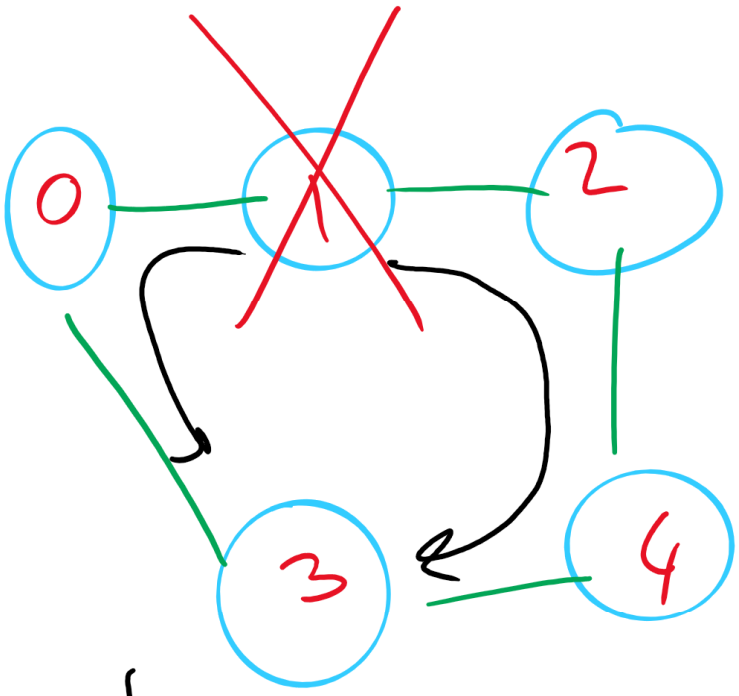
Runtime of BFS/DFS



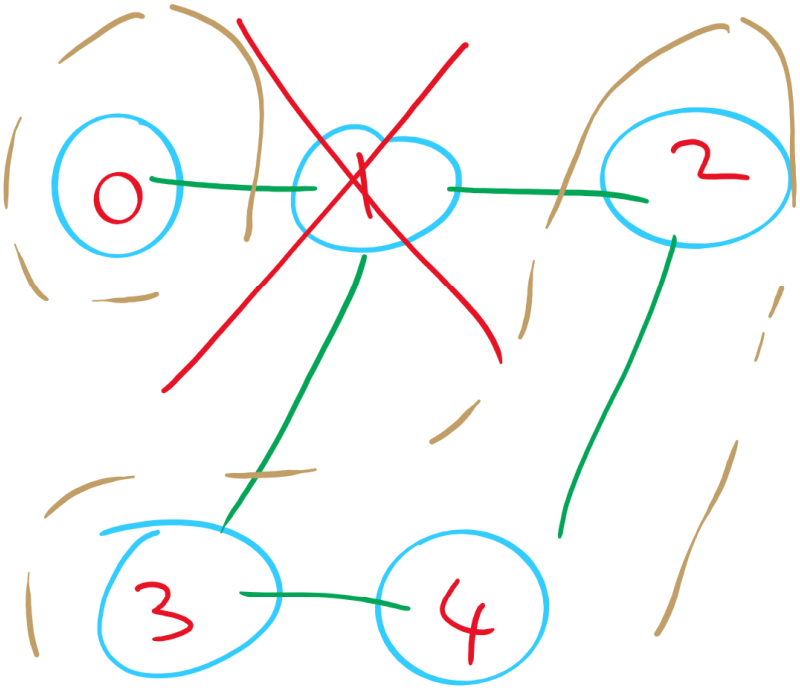
Biconnected graphs

- A *biconnected graph* has at least 2 distinct paths (no common edges or vertices) between all vertex pairs
- Any graph that is not biconnected has one or more *articulation points*
  - Vertices, that, if removed, will separate the graph
- Any graph that has no articulation points is biconnected
  - Thus we can determine that a graph is biconnected if we look for, but do not find any articulation points

Bi-connected Graph



bi-connected

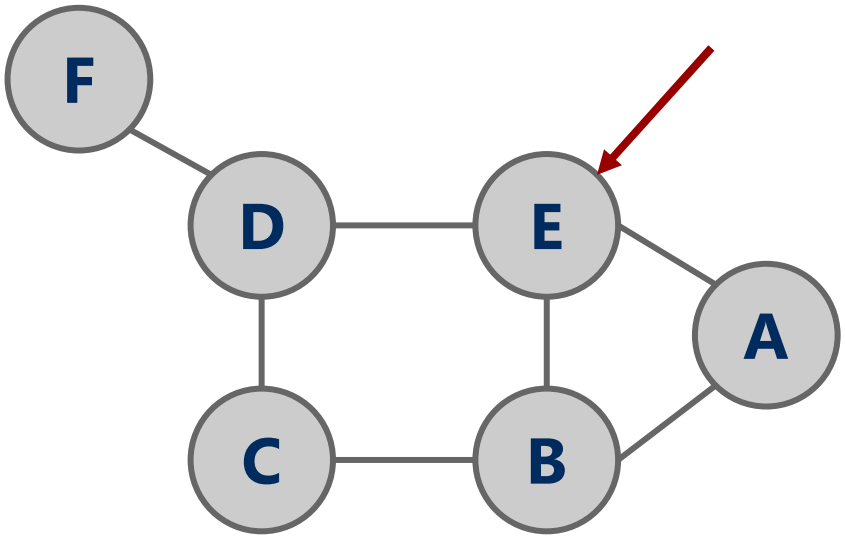


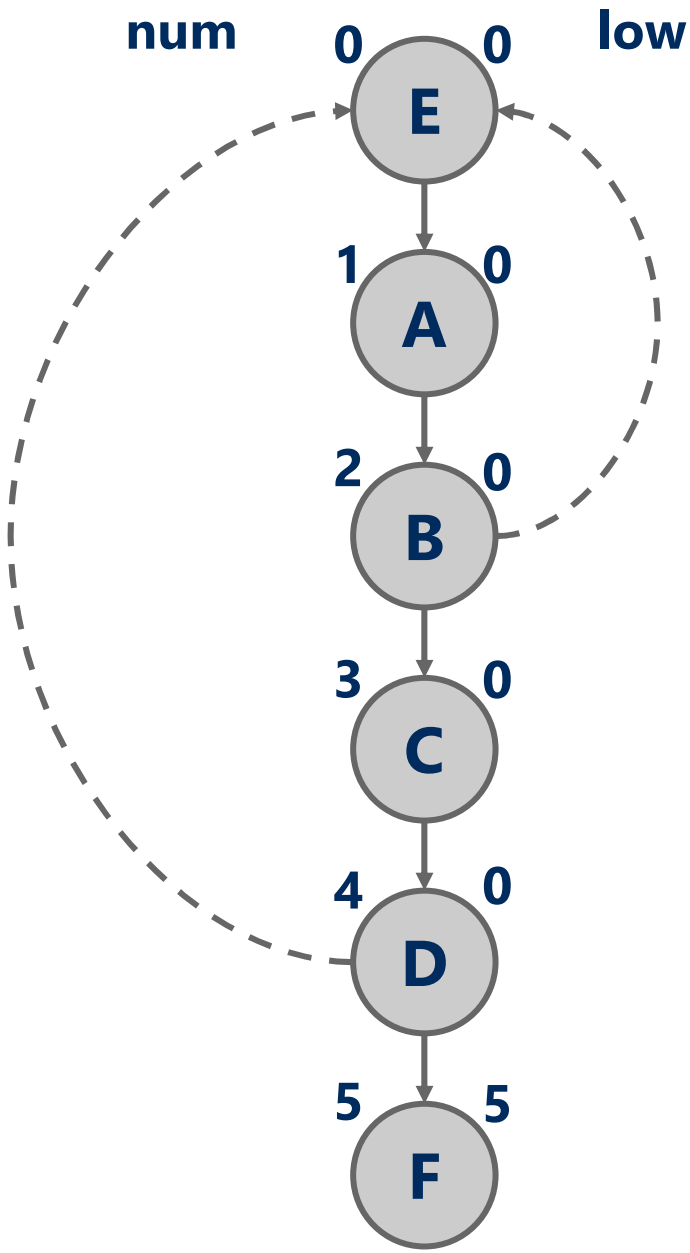
not bi-connected

Finding articulation points

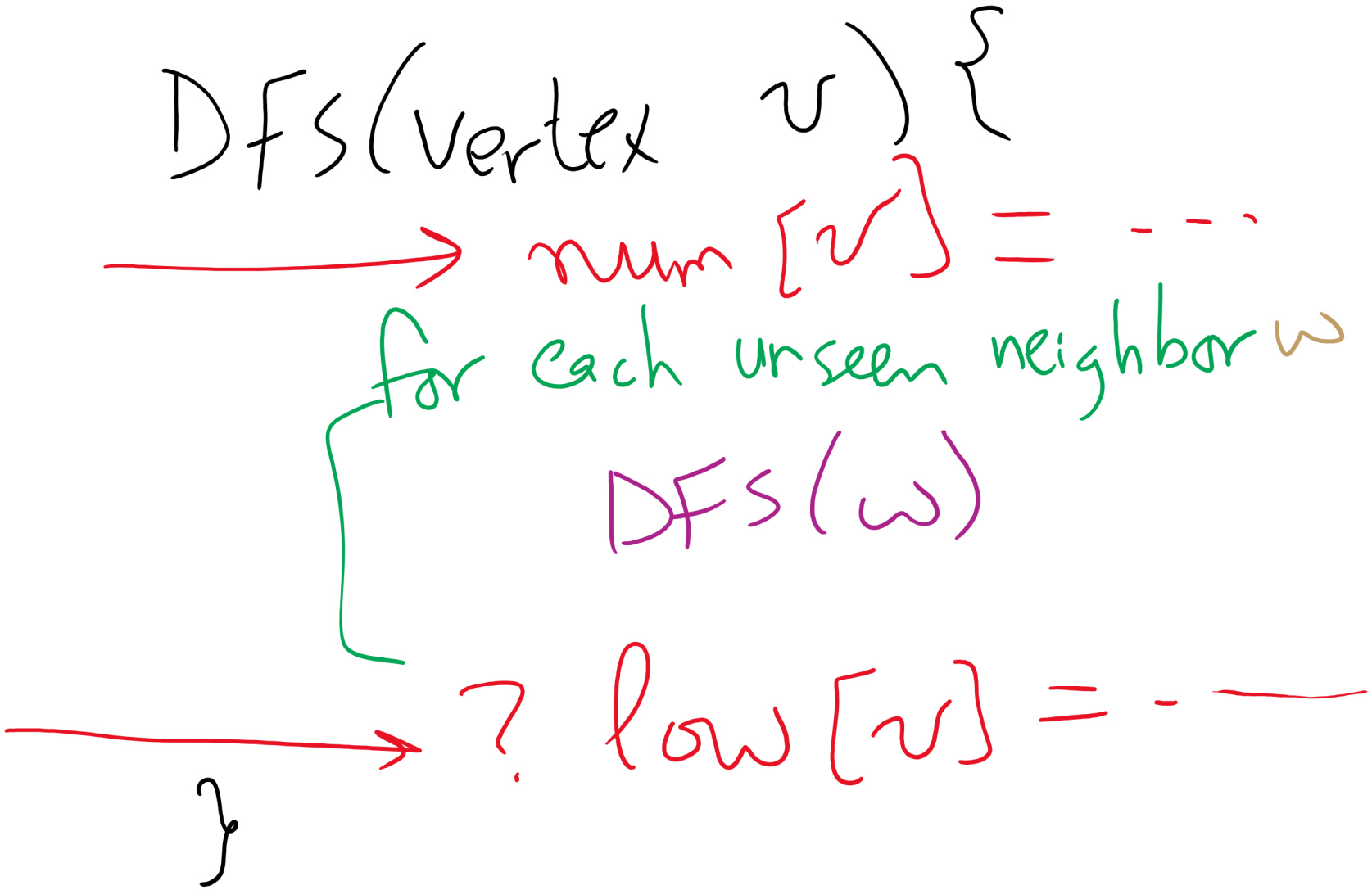
- Variation on DFS
- Consider building up the spanning tree
  - Have it be directed
  - Create "back edges" when considering a vertex that has already been visited in constructing the spanning tree
  - Label each vertex v with with two numbers:
    - num(v) = pre-order traversal order
    - low(v) = lowest-numbered vertex reachable from v using 0 or more spanning tree edges and then at most one back edge
      - Min of:
        - num(v)
        - Lowest num(w) of all back edges (v, w)
        - Lowest low(w) of all spanning tree edges (v, w)

Finding articulation points example

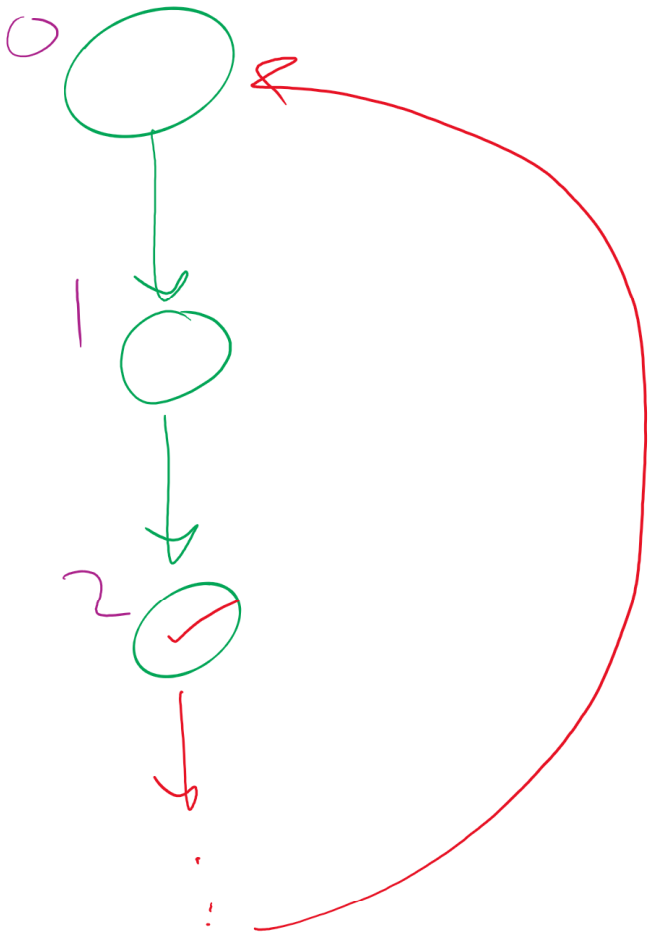




Using DFS to find articulation points



Articulation Points Algorithm



$$low[w] = \min \begin{cases} num[w] \\ num[x] \\ \quad x: \text{back-edge neighbor} \\ low[y] \\ \quad y: \text{child of } w \end{cases}$$

So where are the articulation points?

- If any (non-root) vertex v has some child w such that low(w) ≥ num(v), v is an articulation point
- What about if we start at an articulation point?
  - If the root of the spanning tree has more than one child, it is an articulation point

Problem of the Day (*contd.*)

- **Input**: A file containing LinkedIn Connection information formatted like the following:
  - Account1: Connection1, Connection2, …
  - Account2: Connection1, Connection2, …
  - …
- **Output**: Answer the following questions:
  - Given two LI accounts, how "far" are they from each other?
    - E.g., 1st connection, 2nd connection, etc.
  - Are the accounts in the file all ***connected***?
    - If not, how many ***connected components*** are there?
  - Are there certain accounts that if removed, the remaining accounts become ***partitioned***?
    - These account are called ***articulation points***

Problem of the Day

- **Neighborhood connectivity project**
  - We want to keep a set of neighborhoods connected with the minimum cost possible
- **Input:** A set of neighborhoods and a file with the following format:
  - neighborhood i, neighborhood j, cost of connecting the two neighborhoods
  - …
- **Output:** A set of neighborhood pairs to be connected and a total cost
  - We can go from any neighborhood to any other **(connected)**
  - The total cost should be minimum (i.e., as small as it can be) **(minimal cost)**

Think Data Structures First!

- How can we structure the input in computer memory?
- Can we use Graphs?
- What about the costs? How can we model that?

We said spatial layouts of graphs were irrelevant

- We define graphs as sets of vertices and edges
- However, we'll certainly want to be able to reason about bandwidth, distance, capacity, etc. of the real world things our graph represents
  - Whether a link is 1 gigabit or 10 megabit will drastically affect our analysis of traffic flowing through a network
  - Having a road between two cities that is a 1 lane country road is very different from having a 4 lane highway
  - If two airports are 2000 miles apart, the number of flights going in and out between them will be drastically different from airports 200 miles apart

We can represent such information with edge weights

- How do we store edge weights?
  - Adjacency matrix?
  - Adjacency list?
  - Do we need a whole new graph representation?
- How do weights affect finding spanning trees/shortest paths?
  - The weighted variants of these problems are called finding the *minimum spanning tree* and the *weighted shortest path*

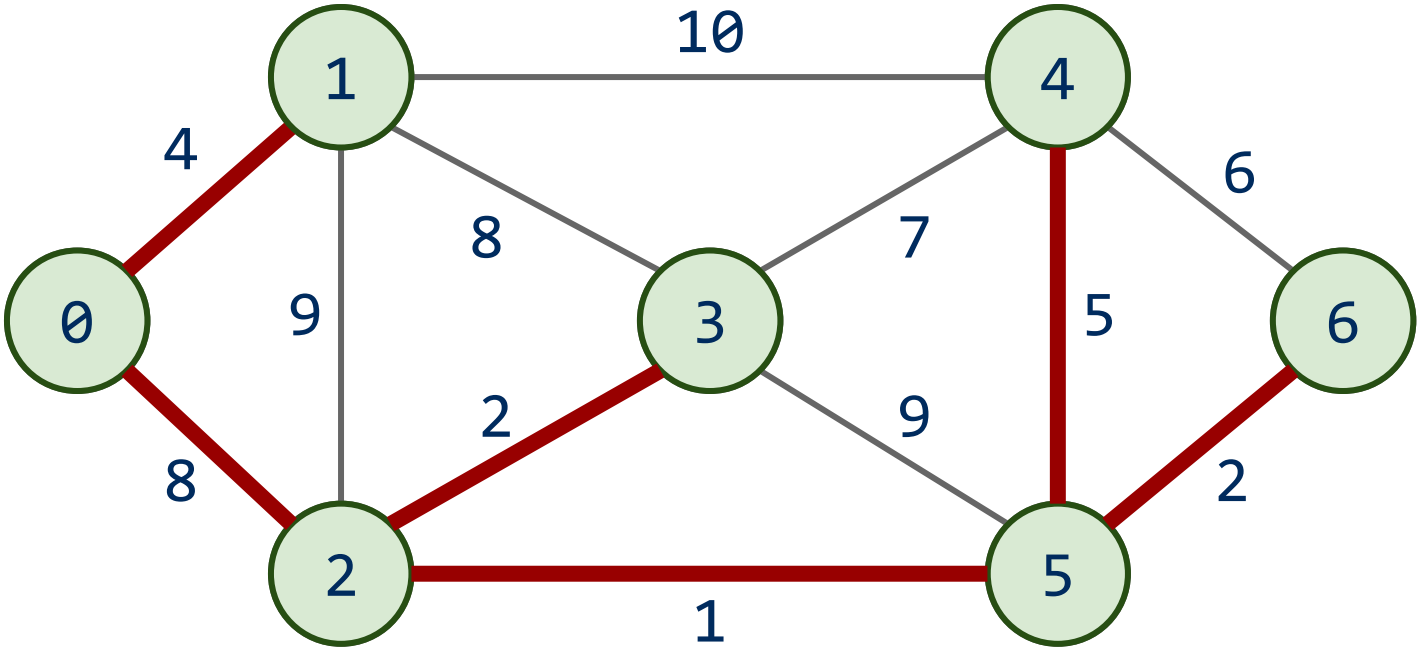
Minimum spanning trees (MST)

- Graphs can potentially have multiple spanning trees
- MST is the spanning tree that has the minimum sum of the weights of its edges

Prim's algorithm

- Initialize T to contain the starting vertex
  - T will eventually become the MST
- While there are vertices not in T:
  - Find minimum edge weight edge that connects a vertex in T to a vertex not yet in T
  - Add the edge with its vertex to T

Prim's algorithm



Please submit your reflections by using the CourseMIRROR App

If you are having a problem with CourseMIRROR, please send an email to coursemirror.development@gmail.com